

Event Documentation

5 October 2006

Copyright 1999-2006, United States Government as represented by the Administrator of the National Aeronautics and Space Administration. No copyright is claimed in the United States under Title 17, U.S. Code.

This software and documentation are controlled exports and may only be released to U.S. Citizens and appropriate Permanent Residents in the United States. If you have any questions with respect to this constraint contact the GSFC center export administrator, <Thomas.R.Weisz@nasa.gov>.

This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD. See <<http://itos.gsfc.nasa.gov/>> or e-mail <itos@itos.gsfc.nasa.gov> for additional information.

You may use this software for any purpose provided you agree to the following terms and conditions:

1. Redistributions of source code must retain the above copyright notice and this list of conditions.
2. Redistributions in binary form must reproduce the above copyright notice and this list of conditions in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD.

This software is provided ‘‘as is’’ without any warranty of any kind, either express, implied, or statutory, including, but not limited to, any warranty that the software will conform to specification, any implied warranties of merchantability, fitness for a particular purpose, and freedom from infringement and any warranty that the documentation will conform to their program or will be error free.

In no event shall NASA be liable for any damages, including, but not limited to, direct, indirect, special or consequential damages, arising out of, resulting from, or in any way connected with this software, whether or not based upon warranty, contract, tort, or otherwise, whether or not injury was sustained by persons or property or otherwise, and whether or not loss was sustained from or arose out of the results of, or use of, their software or services provided hereunder.

Overview of directories and cgi scripts

Overview of classes

Parsed page formats is a description of the output of page_parser.cgi, the input to PageCanvasMap.

Running IselectPage as an application

Running TelemPage as an application

Classes and interfaces:

DPSCConnection

DPSMsgError

DPSThread

DisplayFrame

FontInfo

Ibrowser

IchooseFont

Icolors

Iconvert_T

IselectPage

Iutils

IviewText

MessageDisplay

MneAccessory

MneEntry

Mnemonic

Monitor

PageCanvas

PageCanvasField

PageCanvasMap

PageConversionResult

PageInfo

PageSelector

PointInfo

SizedField

TelemPage

1 Overview of directories and cgi scripts

When a browser is used to display telemetry pages, the following should be in place:

1. A directory (or link) in the parent directory of public_html named "pages". All page files should be in the tree rooted at "pages" and should have the suffix ".page". Code in select_page.cgi uses path "/home/<mission>/pages" to access that directory.
2. A subdirectory (or link) in public_html named "classes" which contains
 - The java class files used for page display.
 - dir_list.cgi
 - page_parser.cgi
 - select_page.cgi
3. A subdirectory (or link) in public_html named "tcvol2". tcvol2 should contain a subdirectory or link named "mnemonics" which contains .html files for mnemonics. These are displayed as help when a user clicks on a mnemonic in a page.

A typical sequence of actions when a browser is used to display telemetry pages is:

1. The user clicks on a link in an html file whose value is something like:


```
dis/select_page.cgi?usr/tcw+trace
```

"/usr/tcw" is the directory containing bin/tcwrn bin/dsp_page_parser.
 "trace" is the mission name.
2. select_page.cgi produces an html file containing an applet with the following parameters:


```
CODE = "IselectPage"
```

CODEBASE = The directory containing select_page.cgi.
 data_port = 7777 (The port number of the data point server.)
 page_parser_url = page_parser.cgi?ARG1+ARG2+/home/ARG2/pages
 Where ARG1 and ARG2 are the arguments to select_page.cgi. ("/usr/tcw" and "trace" in the example above.)
 mnemonics_dir = tcvol2/mnemonics
3. The applet IselectPage starts and displays directories and page files in "pages" by invoking the url dir_list.cgi and writing the output to its "page.list" panel.
4. When the user selects a page file for display, TelemPage and PageCanvas objects are created. The PageCanvas object invokes page_parser.cgi to read the .page file and return a parsed version of that file.

2 Overview of classes

Class `IselectPage` is an applet which can also be run as an application. It displays the following widgets:

- A text area which displays information on the operation currently being carried out.
- A list on the left side of the panel containing page files and subdirectories in the current directory.
- A list on the right side of the panel containing pages currently being displayed.
- A "Display" button.
- An "Up a Directory" button.
- A choice widget with choices "Telemetry Page", "Parsed Page" and "Page Source".

When the user selects a directory name or the "Up a Directory" button, `dir_list.cgi` is invoked and its output is put into the left-side list.

When the user selects a page file name, the actions taken depend on the current contents of the choice widget:

- Telemetry Page: a `TelemPage` object is created and the current directory and the selected page file name are passed to it.
- Parsed Page: An `IviewText` object is created and the url for `page_parser.cgi` is passed to it.
- Page Source: An `IviewText` object is created and the path to the page is passed to it.

Class `TelemPage` is a `Frame` created by `IselectPage` to display a single telemetry page. `TelemPage` can also be run as an application.

`TelemPage` contains the following objects:

- A panel at the top containing buttons "Close", "Stop Data", "Data Server" and "Font".
- A one-line Text Field which displays status messages.
- A `PageCanvas` object which contains all fields of the telemetry page.

Class `PageCanvas` is a `Canvas` which displays all data of a single telemetry page. It contains two major data members:

1. A `PageCanvasMap` object, which stores all information from the page definition file and the current value of all mnemonics on the page.
2. A `DPSCConnection` object, which maintains the socket connection to the data point server and a thread that waits for input over that socket.

When the `DPSCConnection` object receives a mnemonic's value from the data point server, it passes the value to this class, which forwards it to the `PageCanvasMap` object.

All text on the page, including mnemonic values and static text, is written by the `PageCanvasMap` object. This object specifies which graphics context is used by the `PageCanvasMap` and so determines whether writing is done directly to the screen or to an offscreen buffer.

This class decides how big to make itself by asking the PageCanvasMap object how big it is and adding some space for borders. That is done during initialization and when the user changes the font.

Class PageCanvasMap stores all information from the page definition file and the current value of all mnemonics on the page.

The constructor for this class is given the text of a url which invokes page_parser.cgi to parse the page file of interest. The output of that url is read and used to create two lists, a list of Mnemonic objects and a list of static text labels. The list of mnemonics is a fairly complex data structure since a single mnemonic can appear several times on the page with a different format, conversion or color each time.

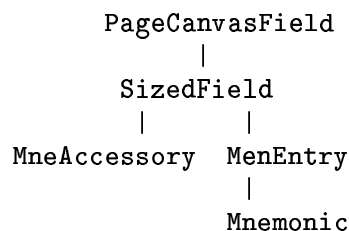
When it is time to paint a field, whether a mnemonic or text field, this class calls the field's paint method.

Class PageCanvasField is the base class for all fields visible on a PageCanvas. A static text field is stored as a PageCanvasField object. Other fields are implemented as objects derived from PageCanvasField.

A PageCanvasField object stores a field's:

- Current value.
- Row and column. (These don't change.)
- Pixel row, column, width and height. (These change when the font changes.)
- Current foreground and background colors.
- An optional link to the next PageCanvasField in a list.

Classes derived from this class are:



SizedField objects add a length (in characters) to objects. The field is always truncated to this length.

An MneAccessory object stores information from an "mne" line in a page file with type l, q, s or t. (See Parsed Page Formats for a description of mne lines.)

An MneEntry object stores information from an "mne" line in a page file with type v.

A Mnemonic object is created from the first "mne ... v" line for each mnemonic.

Class DPSCConnection defines method startDPSCConnection, which opens a socket connection to the data point server and starts a thread which reads all input from that socket. Every time that socket is opened, SHO requests are sent to the data point server for all mnemonics.

This class also defines a method for sending messages to the data point server.

This class maintains a Monitor object, which displays messages between this application and the data point server. A user can make the monitor visible by clicking the "Data server" button in TelemPage.

3 Parsed Page Formats

Before a telemetry page can be displayed, `page_parser.cgi` is invoked with the name of a .page file. It reads the page file and outputs a "parsed" version of that file, which is read by an object of class `PageCanvasMap`. That output can contain three kinds of line: text, mnemonic and conversion.

A **text** line describes a static text field and has the form

```
text <row> <col> <fg> <bg> "<text>"
```

<row> and <col> are integers giving the position of the beginning of the text string in character rows and columns.

<fg> and <bg> give the foreground and background colors of the text. Either or both may be "default".

<text> is the text of the string.

A **mnemonic** line means that information on a mnemonic is displayed. It can have any of the following formats:

```
mne <row> <col> <len> <fg> <bg> <name> v "<raw>"
mne <row> <col> <len> <fg> <bg> <name> v "<raw>" "<cnv>"
mne <row> <col> <len> <fg> <bg> <name> v "<raw>" "<cnv>" "<convName>"
mne <row> <col> <len> <fg> <bg> <name> t "<text>"
mne <row> <col> <len> <fg> <bg> <name> s
mne <row> <col> <len> <fg> <bg> <name> l
mne <row> <col> <len> <fg> <bg> <name> q
```

<len> is the maximum number of characters used to display the mnemonic.

<name> is the mnemonic's name.

<raw> is the format used when the mnemonic's raw value is displayed.

<cnv> is the format used when the mnemonic's converted value is displayed. If this field is not present, the mnemonic's raw value is displayed.

<convName> is the name of a conversion defined in "cnv" lines in this file. When this field is present, that conversion will be applied to the mnemonic's raw value and the result is displayed. When this field is absent, the conversion specified in the database is used.

Lines with type 't' specify a fixed text field. The only difference between these lines and text lines is that a user can click on the text created by one of these lines to get information on the mnemonic.

Lines with type 's' mean that the mnemonic's static flag is displayed as a single character. If the static flag is off, a blank is displayed. Otherwise a "*" is displayed.

Lines with type 'l' mean that the mnemonic's limits flag is displayed as a two-character string, " " for in limits, "RH" for red high, and so on. (See method `MneAccessory.setFlag` for a complete description of strings used for limits.)

Lines with type 'q' mean that the mnemonic's Bad Quality flag is displayed as a single character. If that flag is off, a blank is displayed. Otherwise a "Q" is displayed.

A **conversion** line is part of a conversion definition which can be applied to mnemonics on the page instead of any conversion defined in the operational database. Conversion lines can have these formats:

```
cnv a <name> <n> <c0> ...
cnv d <name> <n> <low> <high> <fg> <bg> "<text>"
```

The "cnv a" format defines an analog conversion.

<name> is the name of the conversion. It can be used as the last field in "mne ... v" lines.

<n> specifies the number of coefficients which follow and can be 1 to 8. The converted value is $c_0 + c_1 * x + c_2 * x^2 + \dots$

where x is the mnemonic's value.

The "cnv d" format defines one state of a discrete conversion. Usually several "cnv d" lines are needed to define the whole discrete conversion.

<n> is the state's number. The highest-numbered state is described first.

<low> and <high> give the state's value range. The mnemonic is in this range if its value is \geq low and $<$ high. When the mnemonic is in this range, <text> is displayed using the colors specified.

Code in `PageCanvasMap.processCnvLine` assumes that the highest-numbered state of a discrete conversion appears before other states for that conversion.

The following lines define a discrete conversion named "lck" with two states:

```
cnv d lck 2 0.000000 0.000000 black green "LOCKED"
cnv d lck 1 1.000000 100.000000 black red "UNLOCKED"
```

4 Running IselectPage as an application

Class IselectPage is the main applet when pages are viewed using a browser. This class can also be run as a java application. It creates a frame listing page files and subdirectories of the root page directory and lets the user navigate the page directory tree and display telemetry pages, page definition files and the parsed version of page definition files.

Before running the application, the environment variable CLASSPATH should be set to include:

1. The directory containing IselectPage.class and the other class files. These files are commonly kept in tcw/dis.
2. The path /usr/local/javalibs/HotJavaBean.jar

When a user left-clicks on a mnemonic field in a page, information on the mnemonic is displayed in a browser. If CLASSPATH does not contain HotJavaBean.jar, that help feature will not work.

4.1 Arguments Accepted by IselectPage

- -host <name>
The application will connect to the data point server on the host named. The default is sunland.gsfc.nasa.gov
- -port <number>
The application will use this port number when connecting to the data point server. The default port number is 7777 and usually will not need to be changed.
- -mission <missionName>
This mission name is passed to page_parser.cgi which assumes that the page directory tree is rooted at /home/<missionName>/pages. The default mission name is trace.
- -dir <directory>
Specifies the directory where the .cgi scripts needed by the application are located. The default directory is /~<missionName>/dis.

4.2 Examples

Using a bash shell, IselectPage could be invoked with these commands:

```
export CLASSPATH=/usr/tcw/dis:/usr/local/javalibs/HotJavaBean.jar
java IselectPage -host sunbaked -mission wire
```

A short description of arguments accepted by the application and their default values can be obtained by typing:

```
java IselectPage -help
```

5 Running TelemPage as an application

Class TelemPage displays a frame containing a single telemetry page and several buttons. Class IselectPage creates one of these objects each time the user requests a telemetry page. It is also possible to run this class as an application to display a single telemetry page.

Before running the application, the environment variable CLASSPATH should be set to include:

1. The directory containing TelemPage.class and the other class files. These files are commonly kept in `tcw/dis`.
2. The path `/usr/local/javalibs/HotJavaBean.jar`

When a user left-clicks on a mnemonic field in a page, information on the mnemonic is displayed in a browser. If CLASSPATH does not contain HotJavaBean.jar, that help feature will not work.

5.1 Arguments Accepted by TelemPage

One argument is required:

- `-page <pagePath>`
Specifies the page to be displayed. `<pagePath>` is the page of the page file starting at the root of the page tree.

Optional arguments:

- `-host <name>`
The application will connect to the data point server on the host named. The default is `sunland.gsfc.nasa.gov`
- `-port <number>`
The application will use this port number when connecting to the data point server. The default port number is 7777 and usually will not need to be changed.
- `-mission <missionName>`
This mission name is passed to `page_parser.cgi` which assumes that the page directory tree is rooted at `/home/<missionName>/pages`. The default mission name is `trace`.
- `-dir <directory>`
Specifies the directory where the `.cgi` scripts needed by the application are located. The default directory is `/~<missionName>/dis`.

5.2 Examples

Using a bash shell, TelemPage could be invoked with these commands:

```
export CLASSPATH=/usr/tcw/dis:/usr/local/javalibs/HotJavaBean.jar
java TelemPage -page acs/acegyro -host sunbaked -mission wire
```

A short description of arguments accepted by the application and their default values can be obtained by typing:

```
java TelemPage -help
```

6 Class DPSCConnection

Exactly one object of this class is created by each PageCanvas. This object establishes a socket connection to the data point server and starts a thread to wait for input from that socket. This class defines a method for sending messages to the data point server which is used by PageCanvasMap when it sends SHO and TAG requests to the data point server.

This class also creates a Monitor object which displays messages to and from the data point server. The user controls whether the monitor is visible and which messages are displayed.

Data members:

```

—
    protected PageCanvas pageCanvas;
    The canvas that created this object
—
    protected String host;
    The data point server's host.
—
    protected int port;
    The data point server's port number.
—
    protected Socket socket = null;
    The socket to the data point server.
—
    protected BufferedReader in;
    Reader on the socket.
—
    protected PrintWriter out;
    Writer to the socket.
—
    protected DPSThread dpsThread = null;
    The thread that reads input from the socket.
—
    protected Monitor monitor;
    A window for displaying messages between this process and the data point server.
—
    protected boolean inCloseDPSCConnection = false;
    Prevents nested calls to closeDPSCConnection.
—

```

```
protected int os;
```

Identifies the operating system for code which is different for different systems. It is set to one of the following values:

```
final int unknown = 0;
final int SunOS = 1;
final int Solaris = 2;
final int Windows95 = 3;
final int Window_16bit = 4;
final int FreeBSD = 5;
```

```
DPSCConnection(String host, int port, PageCanvas pageCanvas)
```

This constructor saves its three arguments. It then calls `System.getProperty("os.name")` to get the name of the operating system and puts it into `os`.

```
public void close()
```

Closes the connection to the data point server, hides and disposes of the monitor window.

```
void closeDPSCConnection()
```

This method:

- Signals `dpsThread` to exit.
- Closes the socket to the data point server.
- Changes the text in the two "Data Server" buttons to "Start data". (One button is in `TelemPage`, the other is in `Monitor`.)

If `inClosedDPSCConnection` is true on entry, this method returns immediately. Otherwise, it sets `inClosedDPSCConnection` to true and then back to false on exit. This is done because this method interrupts `dpsThread`. Sometimes that interrupt causes `dpsThread` to call this method, and the flag `inClosedDPSCConnection` prevents the nested call from proceeding.

```
protected void errorMsg(String msg)
```

Displays `msg` in the frame's status area as an error. Also displays it in the monitor window if its message level is `Connect` or higher.

```
protected void infoMsg(String msg)
```

Displays `msg` in the frame's status area. Also displays it in the monitor window if its message level is `Connect` or higher.

```
public boolean isConnected()
```

Returns true if `socket` is non-null. False otherwise.

```
public boolean toDPS(String msg, int level)
```

Sends `msg` to the data point server and returns true if successful. Also sends `msg` to the monitor with level `level`.

```
public void showMonitor()
```

Makes the monitor visible.

```
void startDPSCConnection()
```

This method attempts to open a socket to the data point server. If successful:

- The text in the two "Data Server" buttons is changed to "Stop data". (One button is in `TelemPage`, the other is in `Monitor`.)

- A DPSThread is started to read input from the socket.
- `pageCanvas.requestAllMne` is called to send SHO requests for all mnemonics to the data point server.

If the socket cannot be opened, an error message is passed to `errorMsg`.

```
protected void warningMsg(String msg)
```

Displays `msg` in the frame's status area as a warning. Also displays it in the monitor window if its message level is `Connect` or higher.

7 Class DPSMsgError

class DPSMsgError

An object of this class is returned by `PageCanvasMap.processValueMsg` to `PageCanvas` when it finds an error in a value message from the data point server.

Data members:

—

String errorMsg;

A string describing the problem.

—

boolean serious = false;

Receiving data for a mnemonic we don't want is not serious. Syntax errors are serious.

DPSMsgError(String msg, boolean serious)

This constructor sets the two data members.

public DPSMsgError set(String msg, boolean serious)

This method sets both data members of an already existing object.

8 Class DPSThread

class DPSThread extends Thread

This class is an inner class of DPSThread.

One instance of this thread is created by DPSThread every time it establishes a socket connection with the data point server. This thread is stopped when that connection is broken. The run method of this thread waits for input from the data point server and processes that input.

The only data member:

protected boolean firstVReceived = false;

is used to detect the first 'v' message from the data point server so a message can be put into the TelemPage's status window.

public void run()

Waits for input from the data point server and processes it:

- If the first character is '?', this is an error message and it is displayed in the monitor and in TelemPage's status window.
- If the message is "ok" it is a reply to a ping and is displayed in monitor.
- If the first character is 'v', this message gives a mnemonic's value. The message is passed to pageCanvas.processValueMsg and displayed in monitor if its display level is Data or higher.
- Otherwise an "unknown" message is displayed in monitor.

This method exits if it finds that DPSThread.dpsThread is not its own thread. Therefore DPSThread signals it to exit by setting dpsThread to null.

protected String getNextMsg()

This method is called by run and returns the next message or null if the connection to the data point server is broken.

This method calls in.readLine to get the next input on the socket from the data point server. That call does not return until input arrives. When code in DPSThread wants to close the socket, that call to readLine must be interrupted and that is done by calling this thread's interrupt method. Unfortunately when Netscape runs java 1.1.2 under Windows 95, there seems to be no way to interrupt a readLine call. Interrupting or even stopping this thread doesn't do it. (This has not been tested under Windows 95 with a newer version of java.)

The solution here is to repeat a poll/sleep cycle until there is something to read before calling readLine when Windows 95 is the operating system.

9 Interface DisplayFrame

```
public interface DisplayFrame
```

This interface is implemented by TelemPage and IviewText and is used by IselectPage.

```
    public void close()
```

This object makes itself invisible and disposes of any resources it is using.

```
    public void show()
```

This object makes itself visible.

10 Class FontInfo

class FontInfo

This class contains information on the font used by all fields on a page. One of these objects is created by each PageCanvas.

Data members:

—

```
String name = "Courier";
int property = Font.PLAIN;
int size = 12;
```

The current font's name, property and size.

—

```
Font font;
```

The current font.

—

```
int charWidth;
```

The width, in pixels, of the character 'E' in the current font.

—

```
int charHeight;
```

The height, in pixels, of a character in the current font with both maximum ascender and maximum descender.

—

```
int rowHeight;
```

The vertical distance, in pixels, between rows with the current font. This equals `charHeight` plus some between-row space.

```
FontInfo(Component comp)
```

This constructor simply passes `comp` to `setConstants`.

```
public void setConstants(Component comp)
```

This method is called after the name, property and/or size of the font have changed. It changes the other fields to match those.

11 Class Ibrowser

see Ibrowser.java

12 Class IchooseFont

See IchooseFont.java

13 Class Icolors

```
public class Icolors
```

This static class defines default foreground and background colors.

Data members are:

```
    static Color default_bg = Color.blue;  
    static String default_bg_str = "blue";  
    static Color default_fg = Color.white;  
    static String default_fg_str = "white";
```

This class does not define any methods.

14 Class Iconvert_T

class Iconvert_T

An object of this class describes a single conversion defined on a telemetry page. Every MneEntry object contains a pointer to an object of this class. When a field does not use a page conversion, that pointer is null. Any number of MneEntry objects can point to a single object of this class.

See parsed page formats for a description of conversion definitions on a parsed page.

Data members:

```

-
    String name;
    The conversion's name
-
    char type;
    static final char analog = 'A';
    static final char discrete = 'D';
    The conversion's type and the two constants it can contain.
-
    float[] coff;
    The coefficients of the polynomial analog conversion. Not used for discrete conversions.
-
    float[] low;
    float[] high;
    Color[] background_color;
    Color[] foreground_color;
    Arrays containing the minimum and maximum values and colors for each state of a
    discrete conversion. Each of these arrays contains max elements. Unused in analog
    conversions.
-
    String[] value;
    An array containing the value for each state of a discrete conversion. Unused in analog
    conversions.
-
    int max;
    In an analog conversion, the number of coefficients in coff. In a discrete conversion, the
    number of discrete states in the conversion.
-
    PageConversionResult conversionResult;
    This is returned by doPageConversion to PageCanvasMap.
    Iconvert_T(String name, char type)
    This constructor sets the name and type fields from its arguments. max is set to -1 and
    conversionResult is allocated.

```

```
void allocArrays(int arrayLen)
```

This method is called by `PageCanvasMap.processCnvLine` when it reads the first line of a discrete conversion. This method sets `max` and allocates arrays `low`, `high`, `background_color`, `foreground_color` and `value`.

```
protected String doAnalog(double val)
```

This method is called by `doPageConversion` to do an analog conversion. It does the conversion on `val` and returns the result.

```
protected void doDiscrete(double val, PageConversionResult result)
```

This method is called by `doPageConversion` to do a discrete conversion. It does the conversion on `val` and puts the result in `result`.

```
public PageConversionResult doPageConversion(String rawStr,  
                                             String rawFormat, String typeStr)
```

This method carries out the conversion on `rawStr`. `rawStr` and `typeStr` are from a value message from the data point server.

If an error is found, a message is put into the `error` field of the object returned. Otherwise, `error` is left null and the converted value is put into the object's `value` field.

```
protected double stringToDouble(String str, String fmt,  
                                PageConversionResult result)
```

This method converts string `str` to its equivalent double using format `fmt` and returns the result. `result.error` is set to null or an error message.

15 Class IselectPage

```
public class IselectPage extends Applet
    implements PageSelector, MessageDisplay
```

This class is the main applet when pages are viewed with a browser and can also be run as an application.

Data members:

—

```
protected Vector displayFrameList = new Vector(4, 4);
```

An array of the current TelemPage and IviewText objects. One of those objects is created each time the user displays a page.

—

```
protected List activeList;
```

A List widget of the names of the current TelemPage and IviewText objects. It is displayed in the right half of the applet. The user can double-click on this list to bring a page to the top. The i'th element in this list is the name of the i'th element in displayFrameList.

—

```
protected TextArea status;
```

The status-display area at the top of the window.

—

```
protected Label dir_label;
```

Displays the pathname of the current directory.

—

```
protected List page_list;
```

Displays the list of page files and subdirectories of the current directory.

—

```
protected Choice option_choice;
```

Lets the user choose whether to display a telemetry page, a page source file or the parsed output of the page parser.

—

```
final String VIEW_TELEM_PAGE = "Telemetry Page";
```

```
final String VIEW_PARSED_PAGE = "Parsed Page";
```

```
final String VIEW_PAGE_SOURCE = "Page Source";
```

Choices available in option_choice.

—

```
protected Button display_button;
```

The "Display" button.

—

```
protected Button up_button;
```

The "Up a Directory" button.

```

-
    final static String default_port_str = "7777";
The default data server port.
-
    protected String base_url_str;
The url of the directory containing the cgi scripts and class files.
-
    protected String page_parser_url_str;
The string used to invoke page_parser.cgi.
-
    protected PageInfo pageInfo;
Packages some of the unchanging information passed to all TelemPage objects.
-
    protected String current_dir_str = "";
The current directory relative to the root page directory. Set to "" when the root is
the current directory.
-
    protected String page_dir = "../pages/";
The path from code base to the base page directory.
-
    boolean isApplet = true;
Set to false when this class is run as an application.
    public void init()
Does initializations needed when run as an applet, then calls commonInit to do initial-
izations needed by both applets and applications.
        void applicationInit(String host, String dir, String port, String mission)
Does initializations needed when run as an application, then calls commonInit to do
initializations needed by both applets and applications.
The four parameters are the arguments entered by the user.
        protected void commonInit(URL url, String data_port_str,
                                String mnemonics_dir_str, String errMsg)
Does initializations common to both applets and applications. These include


- Creating the visible widgets.
- Storing information in a PageInfo object which will be passed to all TelemPage frames.
- Displaying the contents of the root page directory.


        public void appendInfo(String msg)
This method appends msg to the current text in the status area.
        public void display_directory(String dir)
This method gets the contents of directory dir via dir_list.cgi, displays them in page_
list and puts the name of the directory above the list in dir_label.

```

```
public void destroy()
```

Closes any open telemetry pages. This method is called by the browser when it removes the html page containing this applet from its list of active pages.

```
public void error(String msg)
```

This method writes `msg` to the status area and to stdout and sets the background color of the status area to red.

```
public void frameClosed(DisplayFrame frame)
```

This method removes a `TelemPage` or `IviewText` object from `displayFrameList` and removes its name from `activeList`.

This method is part of the `PageSelector` interface.

```
public Applet getApplet()
```

This method is part of the `PageSelector` interface and returns a pointer to this class if it is running as an applet or null if it is running as an application.

This method is called by `PageCanvas` objects when starting a browser to display information on a mnemonic. It is also called by `TelemPage` to determine whether it is part of an applet or application. That's necessary because an "Unsigned Java Applet Window" panel is added to frames in applets and `TelemPage` must allow space for it.

```
public PageInfo getPageInfo()
```

This method is part of the `PageSelector` interface and returns `pageInfo`, a `PageInfo` object.

This method is called by `TelemPage` objects to get global information.

```
public boolean handleEvent(Event evt)
```

This method handles the following events:

- When the application frame menu is used to close the window, `System.exit` is called.
- When the "Up a Directory" button is clicked, `up_dir` is called.
- When The "Display" button is clicked or an item in `page_list` is double-clicked `process_choice` is called.
- When an item in `activeList` is double-clicked, `pop_page` is called.

```
public void newInfo(String msg)
```

This method writes `msg` to the status area and sets the background color of the status area to yellow.

```
public void pop_page()
```

This method is called when the user double-clicks a name in `activeList`. The frame displaying the named page is popped to the top.

```
public void process_choice()
```

This method is called when the user selects a name from `page_list`.

If the name selected ends with a slash, it is assumed to be the name of a subdirectory and is appended to `current_dir_str`. `display_directory` is called to list the contents of the subdirectory.

If the name selected does not end with a slash, it is the name of a page file and is passed to one of the methods `view_telem_page`, `view_parsed_page` or `view_page_source`. The current selection in `option_choice` determines which of those methods is called.

```
public void success(String msg)
```

This method writes `msg` to the status area and sets the background color of the status area to green.

```
public void up_dir()
```

This method is called when the user clicks the "Up a Directory" button. If `current_dir_str` is not empty, this method removes the last directory from `current_dir_str`, then calls `display_directory` to display the new current directory. If `current_dir_str` is already the empty string, a warning is printed.

```
public void view_page_source(String item_str)
```

This method creates a url name for page `item_str` and then creates an `IviewText` object and passes the url name to it.

If the `IviewText` object starts successfully, it is added to `displayFrameList` and the page name is added to `activeList`.

```
public void view_parsed_page(String item_str)
```

This method creates a url name which invokes the page parser on page `item_str`. It then creates an `IviewText` object and passes the url name to it.

If the `IviewText` object starts successfully, it is added to `displayFrameList` and the page name is added to `activeList`.

```
public void view_telem_page(String item_str)
```

This method creates a `TelemPage` object and passes it the current directory and `item_str`, the name of the page to be displayed.

If the `TelemPage` object starts successfully, it is added to `displayFrameList` and the page name is added to `activeList`.

```
public void warning(String msg)
```

This method simply passes `msg` to `error`.

16 Class Iutils

```
public class Iutils
```

This class defines three utility methods used in various places.

```
    public static int conv_bin_str_to_int(String str)
        throws NumberFormatException
```

Converts the binary string `str` to an integer. An exception is thrown if `str` contains characters other than '0' or '1'.

```
    public static Color str_to_color(String str, char type)
```

This method converts the string `str` to a color. Accepted strings are:

- Color names like "red" or "black".
- ITOS color strings from "code_0" to "code_7".
- The string "default". The color returned is the default foreground or background color defined in `Icolors`. If `type` is 'f' the default foreground is returned. If it is 'b' the default background is returned.
- Any other string is treated like "default".

`type` is ignored except in the default cases.

```
    public static int atoi(String s)
```

Converts a decimal, octal or hex string to an integer.

17 Class IviewText

See IviewText.java

18 Interface MessageDisplay

```
public interface MessageDisplay
```

This interface is implemented by classes capable of displaying error, warning and informational messages from other objects. IselectPage and TelemPage currently implement it. TelemPage and IviewText use it.

This interface allows IviewText to pass messages to the frame that created it without assuming that the frame is a TelemPage.

Methods in the interface:

```
public void error(String msg);  
public void warning(String msg);  
public void newInfo(String msg);  
public void appendInfo(String msg);  
public void success(String msg);
```

19 Class MneAccessory

```
class MneAccessory extends SizedField
```

One of these objects is created for each `mne` line in the input received from `page_parser.cgi` with type 'l', 'q', 's' or 't'. These objects are put into the `otherEntries` list of the Mnemonic object for the mnemonic.

See parsed page formats for a description of lines sent by `page_parser.cgi`.

The only data member of this class is

```
char type;
```

which contains one of the letters 'l', 'q', 's', 't' to indicate the type of the `mne` line that created it.

```
MneAccessory(char type)
```

This constructor simply saves `type`.

```
public synchronized void setFlag(String flagStr)
```

`flagStr` is the flag field from a data point server 'v' message. This method uses that string and the object's type to set its value to a one or two-character string:

1. If `type` is 's', `value` is set to "*" if `flagStr` contains an 's' (meaning the mnemonic is static) and to a blank otherwise.
2. If `type` is 'l', `value` is set to a two-character string to reflect the mnemonic's limit status. Two blanks indicates in limits, "RH" indicates red high, and so on.
3. If `type` is 'q', `value` is set to "Q" if `flagStr` contains a 'Q' (meaning the mnemonic has bad quality) and to a blank otherwise.

There is no case in this method for type 't' because text fields don't change value.

20 Class MneEntry

class **MneEntry** extends SizedField

One of these objects is created for each `mne` line in the parsed page with type 'v'. The first `mne` line creates an object of type `Mnemonic` (derived from this class). Subsequent "`mne...v`" lines create objects of this class. These subsequent lines may display the mnemonic using a different format or conversion. The `MneEntry` objects created by these lines are linked into the `otherEntries` list of the `Mnemonic` object.

See parsed page formats for a description of lines in a parsed page.

Data members:

—

`Iconvert_T conversion = null;`

Pointer to a named conversion.

—

`boolean isRaw;`

Specifies whether this field displays a raw or converted value.

`public String setValue(String rawStr, String rawFormat, String cnvStr,
String typeStr, Color fg, Color bg)`

The arguments, except for `rawFormat`, came from a data point server 'v' message. This method assigns a new value to this object based on those arguments.

21 Class Mnemonic

```
class Mnemonic extends MnemonicEntry
```

One of these objects is created from the first "mne ... v" line in the parsed page for each mnemonic. These objects are all linked into the list `mnemonicList`.

Subsequent "mne ... v" lines for the same mnemonic create `MnemonicEntry` objects which are put into the `otherEntries` list of this object.

Mne lines with type other than 'v' create `MnemonicAccessory` objects which are also put into the `otherEntries` list.

It is possible for a parsed page to contain an "mne ... t" line with no "mne ... v" line for the mnemonic. In that case, one of these objects is created but its `rawFormat` field remains null. The same is possible for types l, q or s. This is not common, but it is common for the "v" line to appear after the other lines, so this situation would exist temporarily.

—

```
String name;
```

The mnemonic's name.

—

```
String rawFormat = null;
```

The mnemonic's raw format. All mnemonics have a raw format so this field remains null only in dummy objects.

—

```
String convertedFormat = null;
```

The mnemonic's converted format. This is optional.

—

```
SizedField otherEntries = null;
```

Points to a linked list of objects created by other mne lines in the parsed page with this mnemonic's name. Those objects can be `MnemonicEntry` or `MnemonicAccessory` objects.

—

```
boolean retagged = false;
```

Requesting a mnemonic from the data point server is a two-step process. We first send a SHO request. In response, the data point server sends one 'v' message that gives the mnemonic's value and its type. The type is used to determine what kind of TAG request to send for the mnemonic. When the SHO request is sent, `retagged` is set to false. When the TAG request is sent, `retagged` is set to true.

```
Mnemonic(String name)
```

This constructor just saves the mnemonic's name.

```
public Dimension expand(Dimension container)
```

Checks whether this object or any of its associated fields extends beyond the rectangle `container` and if so, returns a larger rectangle. Otherwise the original rectangle is returned. "Associated fields" are all fields in the `otherEntries` list.

```
public SizedField getField(int x, int y)
```

If this Mnemonic object or any field associated with it contains pixel location x, y, that field is returned. Null otherwise.

```
public String makeSHO()
```

Returns the text of a SHO message to the data point server for this mnemonic.

```
public String makeTAG(String typeStr)
```

Returns the text of a TAG message to the data point server for this mnemonic.

`typeStr` is the type string of a 'v' message from the data point server which set this mnemonic. Non-telemetry time mnemonics are requested at a fixed interval. Others are requested every time their value changes.

```
public void paint(Graphics g, boolean allFields, boolean extraEvent)
```

Paints this mnemonic and any associated fields (that is, fields in the `otherEntries` list). If `allFields` is false, only fields which have changed are painted. If `extraEvent` is true, an extra graphics event is generated after the last paint.

```
public void setPixelValues(FontInfo fi)
```

Sets the x, y, width and height fields for this object and any other fields associated with it. (That is, any fields in the `otherEntries` list.)

```
public String setValues(String rawStr, String cnvStr, String typeStr,  
                        String flagStr, Color fg, Color bg)
```

The arguments came from a data point server 'v' message. This method assigns those values to all fields associated with this mnemonic. Null or an error message is returned.

22 Class Monitor

`class Monitor extends Frame`

One object of this class is created by each `DPSCConnection` object. This class could logically be an inner class of `DPSCConnection`, but static variables are not allowed in inner classes.

An object of this class has a text area for displaying messages to and from the data point server, and buttons for:

- Hiding the frame. (`closeButton`)
- Clearing the text area. (`clearButton`)
- Sending a PING to the data point server. (`pingButton`)
- Making and breaking the connection to the data point server. (`dataButton`)
- Specifying which level of messages are displayed in the text area. (`levelChoice`)

Data members:

—

`protected int msgLevel = Connect;`

Specifies the current message level, which determines which messages are displayed. Its possible values are:

`static final int None = 0;
static final int Connect = 1;
static final int Request = 2;
static final int Data = 3;`

—

`protected Choice levelChoice;`

Allows the user to set `msgLevel`. Its entries are:

`protected final String NoneStr = "No msgs";
protected final String ConnectStr = "connect msgs";
protected final String RequestStr = "request msgs";
protected final String DataStr = "all msgs";`

—

`protected TextArea textArea;`

The area where messages are displayed. This takes up the lion's share of the frame.

—

`protected Button closeButton;`

Closes this frame.

—

`protected Button clearButton;`

Clears the current text from `textArea`.

—

`protected Button pingButton;`

Sends a "ping" message to the data point server.

—

```
protected Button dataButton;
```

Makes or breaks the connection to the data point server.

—

```
protected DPSTConnection dpsConnection;
```

The DPSTConnection object whose messages are being displayed.

—

```
protected String textFont = "Dialog";
protected int textFontSize = 11;
protected String buttonFont = "Helvetica";
protected int buttonFontSize = 14;
```

Fonts for textArea and the buttons.

```
Monitor(DPSTConnection dpsConnection)
```

This constructor creates textArea and the buttons.

```
public boolean handleEvent(Event evt)
```

This method handles clicks on all buttons.

```
protected void msg(String msg, int level)
```

Displays msg in textArea if level is less than or equal to msgLevel.

```
public void setDataButtonText()
```

If dpsConnection.isConnected returns true, the text in dataButton is set to "Stop data". Otherwise it is set to "Start data".

23 Class PageCanvas

class PageCanvas extends Canvas

Exactly one object of this class is created by every TelemPage object to display all data specified by a single page definition file. The two major data members of this class are:

1. **pageMap**, which processes the page file and saves all information from that file. All information about the fields on the page and their current contents resides in this object.
2. **dpsConnection**, which opens a socket to the data point server, reads all data from that socket and passes it to the PageCanvasMap object. Methods for sending data to the data point server also reside in this object.

The classes for those two objects are logically inner classes of this class, but they are put into separate files to keep file sizes reasonable.

There are several times when code in **pageMap** or **dpsConnection** needs a service provided by the other. One way to handle that would have been to give each of the objects a pointer to the other. Instead it was decided to define methods in this class which forward requests between **pageMap** and **dpsConnection**. Those methods are:

- **processValueMsg**
- **requestAllMne**
- **sendToDPS**

Data members:

```

-
    protected String pageName;
    The name of the page definition file.
-
    protected String pagePath;
    The directory containing the page definition file.      This path begins at
    pageInfo.pageRoot.
-
    TelemPage frame;
    The Frame in which created this Canvas and is displaying it.
-
    protected PageCanvasMap pageMap;
    Contains all information from the page definition file.
-
    public DPSConnection dpsConnection = null;
    The connection to the data point server.
-
    protected int height;
    This Canvas's height in pixels.
```

```

—
    protected int width;
This Canvas's width in pixels.
—
    public FontInfo fontInfo;
An object containing information on the font.
—
    public boolean initializing = true;
True from the time this object is created until all mnemonics have been received at
least once, or until the user stops data. This member exists to tell whether it makes
sense to send a message to the page selector's text area. During initialization it does
make sense, but after initialization it would be confusing, so messages are put only in
the frame's status area.
—
    protected boolean useOffscreenImage = true;
Specifies whether the page is saved in an offscreen Image. This reduces flicker when
values change, but uses more memory.
—
    protected Image offscreenImage = null;
The offscreen image where the page is created before being displayed. Used only when
useOffscreenImage is true.
—
    protected Graphics offscreenGraphics = null;
The graphics context for offscreenImage. Used only when useOffscreenImage is true.
—
    protected final int BORDER = 20;
size of the border at the right and bottom of the page in pixels.
    PageCanvas(String pageName, String pagePath, TelemPage frame)
This constructor saves its three arguments, sets the background color of the canvas and
creates fontInfo.
    public void clearOffscreenResources()
This method frees any resources currently used for offscreen storage of the page.
Note that this method does not change "useOffscreen".
    public void close()
This method simply calls dpsConnection.close. It is called by frame when the frame is
closed.
    boolean connect(PageInfo pageInfo)
This method creates dpsConnection and requests it to open a socket to the data point
server.
If the connection is successfully established, true is returned. False otherwise.

```

```
boolean display(PageInfo pageInfo)
```

This method is called immediately after the constructor. It creates the visible part of this object. That includes creating the PageCanvasMap object to parse the page and calculate this canvas's width and height.

If all goes well, true is returned. False Otherwise.

```
public Dimension getSize()
```

Returns width and height of the canvas. This method is not called from ITOS code, but may be called by some java implementations.

```
public boolean handleEvent(Event evt)
```

This method handles only one event, a left click. If the click is on a field associated with a mnemonic, the following actions are taken:

1. The foreground and background colors of the field are reversed.
2. A browser is created to display the mnemonic's help .html file.
3. After a short pause the field's usual foreground and background colors restored.

```
protected void paint(Graphics g)
```

If useOffscreenImage is false, painting is done directly to the screen and this method simply calls pageMap.paint() to draw all current values to "g".

Otherwise, this method:

1. Checks whether offscreen resources have been allocated and calls setOffscreenResources if not. That call also writes current values to the offscreen image.
2. Copies the offscreen image to "g".

```
protected void paint(SizedField field, boolean invert)
```

This method paints a single field. If invert is true, the field's foreground and background colors are reversed.

This method is called by handleEvent when the user clicks on a field associated with a mnemonic. The field is displayed with reversed colors for a short time while the help browser is created.

```
String processValueMsg(String msg)
```

msg is a "v" message just received from the data point server by dpsConnection. This method passes msg to pageMap.processValueMsg. If that method reports a serious error, an error message is displayed in the status area of the frame containing this canvas. If there is no error and an offscreen image is being used, that call to pageMap.processValueMsg put the new value into the offscreen image. This method copies the offscreen image to the screen.

```
void requestAllMne()
```

This method simply calls pageMap.requestAllMne. This method is called by dpsConnection after it has opened a socket to the data point server. At that point we want to send a SHO request to the data point server for all mnemonics on the page. The list of mnemonics resides in pageMap but dpsConnection does not have a connection to pageMap so it sends the request by calling this method.

```
public void sendToDPS(String msg)
```

This method simply passes `msg` to `dpsConnection.sendToDPS`. This method is used by `pageMap` when it wants to send a SHO or TAG message to the data point server.

```
protected void setOffscreenResources()
```

This method is called by `paint()` when it finds that an offscreen image should be used but the offscreen image and graphics context have been allocated.

This method allocates a new image and graphics context, then calls `pageMap.paint()` to draw all current values to the image.

```
protected void setPixelValues()
```

This method:

1. Requests `pageMap` to set pixel values for all fields on the page.
2. Gets the maximum x and y pixel value for any field from `pageMap`.
3. Sets the height and width of the canvas by adding a few pixels (for a border) to the x and y obtained from `pageMap`.

```
public void setSizes()
```

This method is called during initialization and every time the user changes the font. It sets the font and the pixel locations of all fields and the canvas's width and height.

```
protected boolean showMnemonicInfo(String mneName)
```

This method starts a browser to display the .html information for mnemonic `mneName`.

If an applet is running, `AppletContext.showDocument()` is used, otherwise an `Ibrowser` object is created to display the information.

24 Class PageCanvasField

class **PageCanvasField**

This class is the base class for all fields visible on a PageCanvas. That is, all objects in PageCanvasMap's lists `labelList` and `mnemonicList`. A static text field is stored as a PageCanvasField object. Other fields are implemented as objects derived from PageCanvasField.

Data members:

```
—
    String value = null;
    The string to be displayed on the PageCanvas.
```

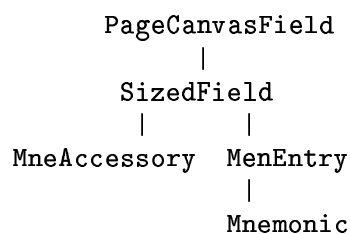
```
—
    int row;
    int col;
    Location of the object in characters.
```

```
—
    int x;
    int y;
    int width;
    int height;
    Location and size of the object in pixels. The top left of the canvas is x = 0, y = 0.
    Storing height seems silly, since all fields on a page have the same height. However, if
    height is not stored here, there must be a pointer to the FontInfo object and that takes
    just as much room. This field cannot be static because it can be different for different
    pages spawned by a single browser.
```

```
—
    Color fg = null;
    Color bg = null;
    Colors for the object.
```

```
—
    PageCanvasField next = null;
    Optional link to another object.
```

Classes derived from this class are:



SizedField objects add a length (in characters) to objects. The field is always truncated to this length.

An **MneAccessory** object stores information from an "mne" line in a page file with type l, q, s or t. (See *Parsed Page Formats* for a description of mne lines.)

An **MneEntry** object stores information from an "mne" line in a page file with type v.

A **Mnemonic** object is created from the first "mne ... v" line for each mnemonic.

```
public boolean contains(int x, int y)
```

Determines whether this object contains pixel location x, y.

```
public Dimension expand(Dimension container)
```

Checks whether this object extends beyond the rectangle **container** and if so, returns a larger rectangle. Otherwise the original rectangle is returned.

```
public synchronized void invert(Graphics g)
```

Paints the object's current value to "g", but reverses its foreground and background colors. This method is synchronized so value or location does not change in the midst of painting.

```
public synchronized void paint(Graphics g)
```

Paints the object's current value to "g". This method is synchronized so value or location does not change in the midst of painting.

```
public synchronized void setPixelValues(FontInfo fi)
```

Sets the object's x, y, height and width fields based on its row, col and value fields and the font constants.

This method is synchronized so **paint** is not called while location fields are changing.

```
public void setXYHeight(FontInfo fi)
```

This method is called by **setPixelValues** to set the object's x, y and height fields.

25 Class PageCanvasMap

class PageCanvasMap

Exactly one of these objects is created by PageCanvas for each telemetry page displayed.

In its constructor, this class invokes `page_parser.cgi` to parse a page definition file, reads its output and stores all information from the page file. That information is stored in two lists. `mnemonicList` stores information on all fields associated with a mnemonic, `labelList` stores information on all text fields.

After initialization, this class receives 'v' messages from the data point server and uses them to update the values of objects in `mnemonicList`.

All screen painting is done by calling `paint` methods of elements of `labelList` and `mnemonicList`.

Data members:

- **protected Mnemonic mnemonicList = null;**

The head of the list of mnemonics.
- **protected PageCanvasField labelList = null;**

The head of the list of text labels. These are not associated with any mnemonic.
- **protected PageCanvas pageCanvas;**

The PageCanvas which created this object.
- **String error = null;**

If the constructor is successful, this member is left null, otherwise it is an error message. PageCanvas tests this member after the constructor returns.
- **int mnemonicsNotReceived = 0;**

The number of mnemonics we have not received from the data point server. This number is initialized when the page file is read and decremented as mnemonics are received. When it reaches zero, `pageCanvas.initializing` is set to false. After that, the value of this member does not matter.
- **private DPSMsgError dpsMsgError;**

This object is passed between various methods. It is not really a class property, but is declared here because we never need more than one at a time and this saves a lot of allocation and freeing.
- **public PageCanvasMap(String url_str, PageCanvas pageCanvas)**

`url_str` invokes `page_parser.cgi` to parse an ITOS page definition file. This constructor invokes that URL and reads its output to build the two lists `mnemonicList` and `labelList`. If any errors are encountered, an error message is put into "error".

`pageCanvas` is the canvas which created this object.

```
protected void addMnemonic(Mnemonic mne)
```

Adds mne to mnemonicList.

```
protected Iconvert_T findConversion(String name, Vector convert_v)
```

Returns the entry for the conversion name from convert_v. Returns null if convert_v does not contain that conversion.

```
public Mnemonic getMnemonic(String name)
```

Returns the Mnemonic object for the mnemonic named name, or null if there is no such mnemonic.

```
public Dimension getTotalRect()
```

This method returns the length and width of the smallest rectangle that contains all fields in labelList and mnemonicList.

PageCanvas calls this method when determining its size.

```
public PointInfo mnemonicAt(int x, int y)
```

Returns information on the mnemonic at pixel location x, y or null if there is no mnemonic at that location.

```
public void paint(Graphics g)
```

This method calls the paint method of all elements of labelList and mnemonicList.

```
protected String parseLine(String line_str, int lineNum,
                           Vector convert_v)
```

This method is called by the constructor each time a line is received from page-parser.cgi. It places information from the line in mnemonicList, labelList or convert_v. If an error is found, an error message is returned. Otherwise null is returned.

```
protected String processCnvLine(String line_str, int lineNum,
                               Vector convert_v)
```

line_str is a "cnv" line received from page-parser.cgi. This method creates an Iconvert_T object and adds it to convert_v.

If an error is encountered, a message is returned. Null otherwise. The argument lineNum is used in error messages.

```
protected String processMneLine(String line_str, int lineNum,
                               Vector convert_v)
```

line_str is an "mne" line received from page-parser.cgi. This method creates a Mnemonic, MneEntry or MneAccessory object depending on the line's type field.

A Mnemonic object is created when the line's type is 'v' and no Mnemonic object exists yet for this mnemonic.

A MneEntry object is created for 'v' lines when there is already one Mnemonic object for this mnemonic.

A MneAccessory object is created when the line's type is 'l', 'q', 's' or 't'.

If a line creates a MneAccessory object for which there is no Mnemonic, a dummy Mnemonic object is created. Dummies are identified by null rawFormat fields. The dummy may be filled in if a 'v' line is received later for this mnemonic.

If an error is encountered, a message is returned. Null otherwise. The argument lineNum is used in error messages.

Argument `convert_v` is an array containing conversions already defined on the page. If this line references a conversion in `convert_v`, the `conversion` field in the `MneEntry` object is pointed to it. Otherwise a dummy conversion with type '?' is created, to be replaced when the conversion definition is read.

```
protected String processTextLine(String line_str, int lineNum)
```

`line_str` is a "text" line received from `page_parser.cgi`. This method creates a `PageCanvasField` object and adds it to `labellist`.

If an error is encountered, a message is returned. Null otherwise. The argument `lineNum` is used in error messages.

```
public DPSMsgError processValueMsg(String msg)
```

`msg` contains an "mne ... v" message from the data point server. This method:

- Extracts information from `msg` and puts it into a `Mnemonic` object.
- If necessary, sends a TAG message to the data point server for this mnemonic.
- Repaints the displayed values which have changed on the canvas.
- If all goes well, null is returned. Otherwise a `DPSMsgError` object is returned.

```
void requestAllMne()
```

This method sends a SHO request to the data point server for each mnemonic. It is called by `DPSCConnection` each time the socket to the data point server is opened.

```
protected String setConversionPointers(Mnemonic mne, Vector convert_v)
```

This method is called by the constructor after all `page_parser.cgi` input has been read. It goes through `mnemonicList` and replaces any pointers to dummy conversions (those with type '?') with pointers to real conversions in `convert_v`.

An error message is returned if there is no definition for a conversion.

```
protected String setOneConversionPointer(MneEntry entry, Vector convert_v)
```

This method is called by `setConversionPointers` for each `MneEntry` in the table `mnemonicList`. If `entry` has a pointer to a dummy conversion (one with type '?'), this method finds the real conversion definition in `convert_v` and substitutes that pointer.

An error message is returned if `convert_v` does not contain a definition for the conversion. Null is returned otherwise.

```
public void setPixelValues(FontInfo fi)
```

This method goes through `labellist` and `mnemonicList` and uses sizes in `fi` to set the `x`, `y`, `width` and `height` fields.

26 Class PageConversionResult

class PageConversionResult

An object of this class contains the results of a page conversion carried out on a value from the data point server. `Iconvert_T.doPageConversion` returns one of these objects to `PageCanvasMap`.

Data members:

—

String value = null;

The converted value.

—

Color fg = null;

Color bg = null;

The colors to use when displaying `value`. Either or both may be null, meaning the default color is used.

—

String error = null;

An error message. If non-null, the other fields should be ignored.

public void clear()

Sets all fields of this object to null.

27 Class PageInfo

class PageInfo

An object of this class is passed from a PageSelector to every TelemPage it creates. The object contains global information the PageSelector probably got from .html file parameters or command line arguments.

Data members:

—

URL baseUrl = null;

A URL created from the path to the directory containing the page parser. This URL is typically created from a string something like:

`http://sunland.gsfc.nasa.gov/~trace/dis/`

—

String pageParserUrlStr = null;

The string used to invoke the page parser. This string typically comes from an applet's html file and looks something like:

`page_parser.cgi?/usr/tcw+trace+/home/trace/pages`

—

String mnemonicsDir = null;

The directory where .html files describing mnemonics are found. This directory is assumed to be a sibling to baseUrl. It is usually `tcvol2/mnemonics`.

—

String pageRoot = null;

The root of the directory tree where page definition files are stored. This string gives that root relative to baseUrl.

—

int port = -1;

The port number of the server socket opened by the data point server.

28 Interface PageSelector

```
public interface PageSelector
```

This interface is implemented by classes which start TelemPage or IviewText objects.

```
public void frameClosed(DisplayFrame frame)
```

This method informs the PageSelector that `frame` is exiting.

```
public PageInfo getPageInfo()
```

This method allows the caller to get the global information contained in a `PageInfo` object.

```
public Applet getApplet()
```

If the program is running as an applet, an `Applet` is returned. If it is running as an application, null is returned.

This method allows the caller to get the Applet pointer for things like getting an Applet-Context. It is also used simply to determine whether an applet or application is running.

29 Class PointInfo

class PointInfo

An object of this class is returned by `PageCanvasMap.mnemonicAt` to `PageCanvas` when the user clicks on the canvas. If the click was on a field associated with a mnemonic, the object contains the mnemonic's name and a pointer to the `SizedField` object representing the field.

Data members:

—

String mnemonicName;

The name of the mnemonic associated with the point.

—

SizedField field;

The field the user clicked.

PointInfo(String name, SizedField fld)

This constructor sets the two data members.

30 Class SizedField

abstract class **SizedField** extends PageCanvasField

Data members:

—

int len;

All fields on a page have a length except static text fields not associated with a mnemonic. (That is, all fields except those defined in "text" lines.)

—

boolean changed = false;

Used to tell whether this field has changed since it was last painted.

Classes MneEntry, Mnemonic and MneAccessory are derived from this class.

public synchronized void assign(String newValue)

Puts **newValue** into this object's **value** field after possibly truncating it to **len** characters. If this action changes the object's **value**, **changed** is set to true.

public void paint(Graphics g)

This method calls PageCanvasField.**paint** to paint the object, then sets **changed** to false.

public synchronized void setPixelValues(FontInfo fi)

This method calls PageCanvasField.**setXYHeight** to set the object's **x**, **y** and **height** fields. It then sets **width** to **len** times the width of a character.

31 Class TelemPage

`class TelemPage` extends `Frame` implements `MessageDisplay`, `DisplayFrame`

An object of this class is a frame which displays one telemetry page. Its major components are:

- A `PageCanvas` object which displays the page contents.
- A one-line status display.
- The buttons:
 - **Close** closes the page permanently.
 - **Stop Data** closes the socket connection between this page and the data point server. When this button is clicked, its text changes to **Start Data** and the next click will restore the connection to the data point server.
 - **Data Server** creates a monitor frame displaying messages to and from the data point server. The user can specify which messages are displayed.
 - **Font** allows the user to change the font size used on the page. When the font is changed, the size of the frame displaying the page also changes.

An object of this class is created by `IselectPage` each time the user requests a page. The class can also be run as an application.

Data members;

```

—
    protected String pageName;
    The name of the page definition file.
—
    protected String pagePath;
    The path to the directory containing the page definition file. This path begins at
    pageInfo.pageRoot.
—
    PageSelector pageSelector = null;
    When this class is run as an application, this member remains null. Otherwise it
    points to the PageSelector which created the class and is used to ask the PageSelector
    for information and to inform it when this page closes.
—
    MessageDisplay msgDisplay = null;
    Used only when there is an object to accept messages from this object.
—
    PageInfo pageInfo = null;
    Information on the page being displayed. Normally this is set in begin() by calling
    pageSelector.getPageInfo. Its only use is in begin. It is a class variable only so it can
    be set by main when run as an application.
—

```

```
protected Button closeButton;
```

The button that closes this page permanently.

—

```
protected Button dataControlButton;
protected final String startString = "Start Data";
protected final String stopString = "Stop Data ";
```

The button that temporarily stops and starts data from the data point server.

—

```
protected Button dpsButton;
```

The button that opens a window showing messages between this object and the data point server.

—

```
protected Button fontButton;
```

The button that allows the user to change the font used in the page canvas area.

—

```
protected TextField status;
```

The field below the buttons giving status information.

—

```
protected PageCanvas pageCanvas;
```

The canvas area where page contents are displayed.

—

```
protected final String VERSION = "        VERSION 2.0 (6-24-98)";
```

The version displayed at the top of the frame.

```
TelemPage(String pagePath, String pageName, PageSelector sel,
          MessageDisplay display)
```

This constructor is used by PageSelectors like IselectPage. It simply saves the four arguments.

```
TelemPage(String pagePath, String pageName)
```

This constructor is used when this class is run as an application. It saves its two arguments and leaves pageSelector and msgDisplay null.

```
boolean begin()
```

This method is always called immediately after one of the constructors. It does the following:

- Insures that a PageInfo object is available.
- Creates the buttons and status line.
- Creates a PageCanvas object and requests it to parse the page file, connect to the data point server and start the thread which will wait for input on that connection.
- Requests information from pageCanvas on its size and sets the size of this frame.
- Makes this frame visible.

If all goes well, true is returned, false otherwise.

```
protected void appendInfo(String msg)
```

This method displays `msg` in the frame's status area with a green background. If `msgDisplay` is not null and `pageCanvas` is still initializing, `msg` is passed to `msgDisplay.appendInfo`.

```
protected void changeFont()
```

This method is called by an `IchooseFont` object when the user saves a font change. It informs `pageCanvas` that the font has changed and then resizes this frame.

```
protected void close()
```

This method is called by `userClosed` when the user closes this page directly and by `pageSelector`, `PageSelector` when it closes all pages.

This method closes the socket to the data point server (if it is open), stops the data point server thread (if it is running) and destroys all windows.

If this class is running as an application, `System.exit` is called.

```
protected void error(String msg)
```

This method displays `msg` in the frame's status area with a red background. If `msgDisplay` is not null and `pageCanvas` is still initializing, `msg` is passed to `msgDisplay.error`.

```
public boolean handleEvent(Event evt)
```

This method handles the following events:

- The user closes the frame. `userClosed` is called.
- The `dataControlButton` button is pressed. `pageCanvas` is instructed to open or close the socket to the data point server depending on the current label of `dataControlButton`.
- `dpsButton` is pressed. `pageCanvas.dpsConnection.showMonitor` is called to display the monitor window.
- The `fontButton` is pressed. An `IchooseFont` object is created and placed over the `fontButton`.

```
protected void newInfo(String msg)
```

This method displays `msg` in the frame's status area with a green background. If `msgDisplay` is not null and `pageCanvas` is still initializing, `msg` is passed to `msgDisplay.newInfo`.

```
protected void setDataControlButton(boolean connected)
```

This method is called by the `DPSCConnection` when the connection to the data point server is established or broken. `connected` specifies whether the connection is now open or closed.

This method changes the text in the `dataControlButton` to show what will happen when that button is pushed.

```
protected void setSize()
```

This method is called by `begin` and each time the user changes the font. It sets the size of this frame by getting the size of `pageCanvas` and increasing the height to allow room for the buttons and status display. If this frame is part of an applet, room must also be provided for the panel at the bottom saying "Unsigned Java Applet Window".

```
protected void success(String msg)
```

This method displays `msg` in the frame's status area with a green background. If `msgDisplay` is not null and `pageCanvas` is still initializing, `msg` is passed to `msgDisplay.success`.

```
protected void userClosed()
```

This method is called when the user closes the window by either the "Close" button or the frame menu.

If `pageSelector` is non-null, `pageSelector.frameClosed` is called to inform the page selector that this page is exiting. Finally, `close` is called to break the connection with the data point server and to dispose of all windows.

```
protected void warning(String msg)
```

This method displays `msg` in the frame's status area with a yellow background. If `msgDisplay` is not null and `pageCanvas` is still initializing, `msg` is passed to `msgDisplay.warning`.